Managing Security in FPGA-Based Embedded Systems

Ted Huffmire
Naval Postgraduate School

Brett Brotherton
Special Technologies Laboratory

Timothy Sherwood
University of California, Santa Barbara

Ryan Kastner
University of California, San Diego

Timothy Levin, Thuy D. Nguyen, and Cynthia Irvine
Naval Postgraduate School

FPGAs combine the programmability of processors with the performance of custom hardware. As they become more common in critical embedded systems, new techniques are necessary to manage security in FPGA designs. This article discusses FPGA security problems and current research on reconfigurable devices and security, and presents security primitives and a component architecture for building highly secure systems on FPGAs.

Because FPGAs can provide a useful balance between performance, rapid time to market, and flexibility, they have become the primary source of computation in many critical embedded systems.[1,2] The aerospace industry, for example, relies on FPGAs to control everything from the Joint Strike Fighter to the Mars Rover. Face recognition systems, wireless networks, intrusion detection systems, and supercomputers, all of which are employed in large security applications, also use FPGAs. In fact, in 2005 alone, an estimated 80,000 different commercial FPGA design projects began.[3]

Because major IC manufacturers outsource most of their operations,[4] IP theft from a foundry is a serious concern. FPGAs provide a viable solution to this problem because the sensitive IP is not loaded onto the device until after it has been manufactured and delivered, making it harder for adversaries to target a specific application or user. Furthermore, modern FPGAs use bitstream encryption and other methods to protect IP once it is loaded onto the FPGA or external memory.

However, techniques beyond bitstream encryption are necessary to ensure FPGA design security. To save time and money, FPGA systems are typically cobbled together from a collection of existing computational cores, often obtained from third parties. These cores can be subverted during the design phase, by tampering with the tools used to translate the design to the cores or by tampering with the cores themselves. Building every core and tool from scratch is not economically feasible in most cases, and subversion can affect both third-party cores and cores developed in-house. Therefore, embedded designers need methods for securely composing systems comprising both trusted and untrusted components.

Reconfigurable systems

Several examples of FPGA applications can help illustrate the utility of FPGAs, along with the need for increased security. We choose encryption, avionics, and computer vision examples because these applications demand high throughput and strong security. We also provide background on FPGA architecture and design flows to review the nuts and bolts of this useful technology.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **2008**	2. REPORT TYPE	3. DATES COVERED **00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE **Managing Security in FPGA-Based Embedded Systems**	5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Department of Computer Science,Monterey,CA,93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S) 11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

IEEE Design and Test of Computers, Vol 25, No. 6, pp 590-598

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**11**	

Motivating examples

FPGAs are a natural platform for the implementation of cryptographic algorithms, given the large number of bit-level operations required in modern block ciphers. Because transformations also require shifting or permuting bits, these operations can be wired into the FPGA, thus incurring extremely low overhead, and with parallelism where appropriate. FPGA-based implementations of MD5, SHA-2, and various other cryptographic functions have exploited this sort of bit-level operation. Even public-key cryptographic systems have been built atop FPGAs. Similarly, there are various FPGA-based intrusion-detection systems (IDS).

All this work centers around exploiting FPGAs to speed cryptographic or intrusion-detection primitives, but it is not concerned with protecting the FPGAs themselves. Researchers are just now starting to realize the security ramifications of building such systems around FPGAs.

Cryptographic systems such as encryption devices require strong isolation to segregate plaintext (red) from ciphertext (black). Typically, red and black networks (as well as related storage and I/O media) are attached to the device responsible for encrypting and decrypting data and enforcing the security policy; this policy ensures that unencrypted information is unavailable to the black network.

In more concrete terms, Figure 1 shows an embedded system with its components divided into two domains, which we have illustrated with different shading. One domain consists of MicroBlaze0 (a processor), an RS-232 interface, and a distinct memory partition. The other domain consists of MicroBlaze1, an Ethernet interface, and another distinct partition of memory. Both domains share an AES (Advanced Encryption Standard) encryption core, and all the components are connected over the on-chip peripheral bus (OPB), which contains policy enforcement logic to prevent unintended information flows between domains. An authentication function to interpret data from a biometric iris scanner (which might be attached to the RS-232 port) could be added to such a layout. However, if the authentication required a high degree of trustworthiness, the implementation of the function would need to reside in a (new or existing) trusted core.

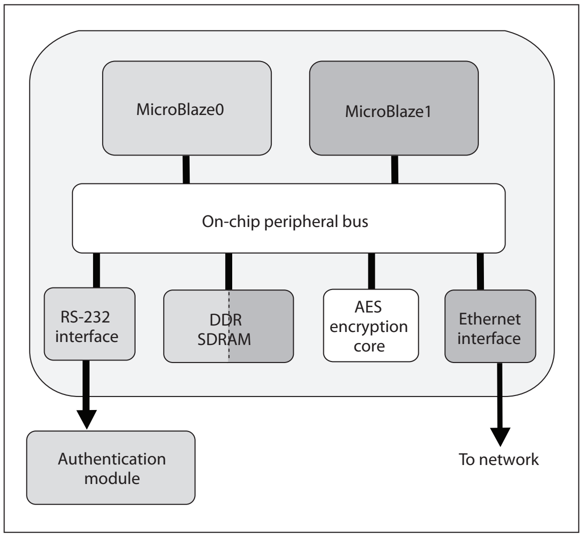

Figure 1: A system consisting of two processors, a shared AES (Advanced Encryption Standard) encryption core, an Ethernet interface, and RS-232 interface, and a shared external DRAM (dynamic RAM), all connected over a shared bus. (DDR: double data rate; SDRAM: synchronous DRAM.) (Revised from T. Huffmire et al., “Designing Secure Systems on Reconfigurable Hardware,” *ACM Trans. Design Automation of Electronic Systems (TODAES), vol. 13, no. 3, July 2008, article 44. ©2008 ACM with permission.*[5])

In the aviation field, both military and commercial sectors rely on commercial off-the-shelf (COTS) FPGA components to save time and money. In military aircraft, sensitive targeting data is processed on the same device as less-sensitive maintenance data. Also, certain processing components are dedicated to different levels of data in some military hardware systems. Because airplane designs must minimize weight, it is impractical to have a separate device for every function or level. Allocation of functions to provide separation of logical modules is a common practice in avionics to resolve this problem and to provide fault tolerance–for example, if a bullet destroys one component.

Lastly, intelligent video surveillance systems can identify potentially suspicious human behavior and bring it to the attention of a human operator, who can make a judgment about how to respond. Such systems rely on a network of video cameras and embedded processors that can encrypt or analyze video in real time using computer vision technology, such as human behavior analysis and face recognition. FPGAs are a natural choice for any streaming application because they can provide deep computation pipelines, with no shortage of parallelism. Implementing such a system would require at least three cores on the FPGA: a video interface for decoding the video stream, an encryption or computer vision mechanism for processing the video, and a network interface for sending data to a security guard's station. Each of these modules must be isolated to prevent sensitive information from being shared between modules improperly–for example, directly between the video interface and the network.

FPGA architecture

FPGAs use programmability and an array of uniform logic blocks to create a flexible computing fabric that can lower design costs, reduce system complexity, and decrease time to market, using parallelism and hardware acceleration to achieve performance gains. The growing popularity of FPGAs has forced practitioners to begin integrating security as a first-order design consideration, but the resource-constrained nature of embedded systems makes it challenging to provide a high level of security.

An FPGA is a collection of programmable gates embedded in a flexible interconnect network that can contain several hard or soft microprocessors. FPGAs use truth tables or lookup tables (LUTs) to implement logic gates, flip-flops for timing and registers, switchable interconnects to route logic signals between different units, and I/O blocks for transferring data into and out of the device. A circuit can be mapped to an FPGA by loading the LUTs and switch boxes with a configuration, a method that is analogous to the way a traditional circuit might be mapped to a set of AND and OR gates.

An FPGA is programmed using a bitstream. This binary data, loaded into the FPGA through specific I/O ports on the device, defines how the internal resources are used for performing logic operations. (For a detailed discussion of the architecture of a modern FPGA, see the survey by Compton and Hauck.[1])

Design flow

Figure 2 shows some of the many different design flows used to compose a single modern embedded system. The FPGA implementation relies on several sophisticated software tools created by many different people and organizations. Special-purpose processing cores, such as an AES core, can be distributed in the form of the hardware description language (HDL), netlists (which are a list of logical gates and their interconnections), or a bitstream. These cores can be designed by hand, or they can be automatically generated by design tools. For example, the Xilinx Embedded Development Kit (EDK) generates a soft microprocessor on which C code can be executed. There are even tools that convert C code to HDL, including Mentor Graphics Catapult C and Celoxica.

An example of an especially complex design flow is AccelDSP, which first translates Matlab algorithms into HDL; logic synthesis then translates this HDL into a netlist. Next, a synthesis tool uses a place-and-route algorithm to convert this netlist into a bitstream, with the final result being an implementation of a specialized signal-processing core. Security vulnerabilities can be introduced into the life cycle inadvertently because designers sometimes leave "hooks" (features included to simplify later changes or additions) in the finished design. In addition, the life cycle can be subverted when engineers inject unintended functionality, some of which might be malicious, into both tools and cores.

Reconfigurable security problems

Design-tool subversion, composition, trusted foundries, and bitstream protection are problems

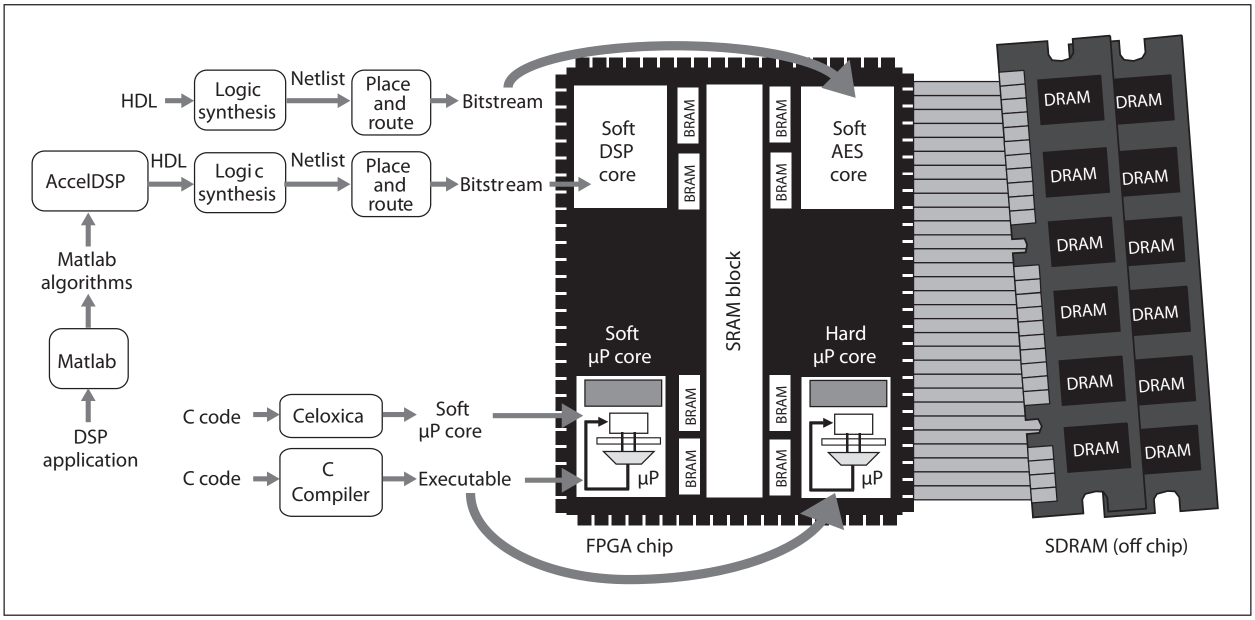

Figure 2: A modern FPGA-based embedded system in which distinct cores with different pedigrees and various trust requirements occupy the same silicon. Reconfigurable logic, hardwired soft-processor cores, SRAM (static RAM) blocks, and other soft IP cores all share the FPGA and the same off-chip memory. (BRAM: block RAM; DSP: digital-signal processing; HDL: hardware description language; μP: microprocessor.)

often associated with reconfigurable hardware.

Design-tool subversion

The subversion of design tools could easily result in malicious design being loaded onto a device. For example, a malicious design could physically destroy the FPGA by causing the device to short circuit. In fact, major design-tool developers have few or no checks in place to ensure that attacks on specific functionality are not included. However, we are not proposing a method that makes possible the use of subverted design tools to create a trusted core. Rather, our methods make it possible to safely combine trusted cores, developed with trusted tools (perhaps using in-house tools that might not be fully optimized) with untrusted cores. FPGA manufacturers such as Xilinx provide signed cores that embedded designers can trust. Freely available cores obtained from sources such as OpenCores might have vulnerabilities introduced after distribution from the original source. However, a digital signature does not prevent a vulnerability either.

The composition problem

Given that different design tools produce a set of interoperating cores, and in the absence of an overarching security architecture, you can only trust your final system as much as you trust your least-trusted design path.[6] If there is security-critical functionality (such as a unit that protects and operates on secret keys), there is no way to verify that other cores cannot snoop on it or tamper with it.

One major problem is that it's now possible to copy hardware, not just software, from existing products, and industry has invested heavily in mechanisms to protect IP. Few researchers have begun to consider the security ramifications of compromised hardware.

Industry needs a holistic approach to manage security in FPGA-based embedded-systems design. Systems can be composed at the device, board, and network levels. At the device level, one or more IP cores reside on a single chip. At the board level, one or more chips reside on a board. At the network level, multiple boards are connected over a network. These multiple scales of design present different potential avenues for attack. Attacks at the device level can involve malicious software as well as sophisticated sand-and-scan techniques. Attacks at the board level can involve passive snooping on the wires that connect chips and the networks that connect boards as well as active modification of data traffic. There are security advantages to using a separate chip for each core, because doing so eliminates the threat of cores on the same device interfering with one another. This advantage must be weighed against the increased power and area cost of having more chips and the increased risk of snooping on the communication lines between chips.

Composing secure system using COTS components also presents difficulties:

- Did the manufacturer insert unintended functionality into the FPGA fabric? Was the device tampered with en route from the factory to the consumer?
- Does one of the cores in the design have a flaw (intentional or otherwise) that an attacker could exploit? Have the design tools been tampered with?
- Does a security flaw exist in the software running on general-purpose CPU cores or in the compiler used to build the software?
- If an embedded device depends on other parts of a larger network (wired or wireless) of other devices (a system of systems), are those parts malicious?

We propose a holistic approach to secure system composition on an FPGA that employs many different techniques, both static and runtime, including life-cycle management, reconfigurable mechanisms, spatial isolation, and a coherent security architecture. A successful security architecture must help designers manage system complexity without requiring all system developers to have complete knowledge of the inner workings of all hardware and software components, which are far too complex for complete analysis. An architecture that enables the use of both evaluated and unevaluated components would let us build systems without having to reassess all the elements for every new composition.

The trusted-foundry problem

FPGAs provide an important security benefit over ASICs. When an ASIC is manufactured, the sensitive design is transformed from a software description to a hardware realization, so the description is exposed to the risk of IP theft. For sensitive military content, this could create a national security threat. Trimberger explains how FPGAs address the problem for the fabrication phase, but the security problem of preventing the design from being stolen from the FPGA itself remains and is similar to that of an ASIC.[7]

Bitstream protection

Most prior work relating to FPGA security focuses on preventing IP theft an securely uploading bitstreams in the field. Because such theft directly impacts the "bottom line," industry has already developed several techniques to combat FPGA IP theft, such as encryption, fingerprinting, and watermarking. However, establishing a root of trust on a fielded device is challenging because it requires incorporating a decryption key into the finished product. Some FPGAs can be remotely updated in the field, and industry has devised secure hardware update channels that use authentication mechanisms to prevent a subverted bitstream from being uploaded. These techniques were developed to prevent an attacker from uploading a malicious design that causes unintended functionality. (Trimberger provided a more extensive overview of bitstream protection schemes.[7])

Reconfigurable security solutions

Solutions to reconfigurable security problems fall into two categories: life-cycle management and a secure architecture.

Life-cycle management

Clearly, industry needs an approach to ensure the trustworthiness of all the tools involved in the complex FPGA design flow. Industry already deals with this life-cycle management problem with software configuration management, which covers operating systems, security kernels, applications, and compilers. Configuration management stores software in a repository and assigns it a version number. The reputation of a tool's specific version is based on how extensively is has been evaluated and tested, the extent of its adoption by practitioners, and whether it has a history of generating output with a security flaw. The rationale behind taking a snapshot in time of a particular version of a tool is that later versions of the tool might be flawed. For example, because automatic updates can introduce system flaws, it is often more secure to delay upgrades until the new version has been thoroughly tested.

A similar strategy is needed for life-cycle protection of hardware to provide accountability in the development process, including control of the development environment and tools, as well as trusted delivery of the chips from the factory. Both cores and tools should be placed under a configuration management system. Ideally, it should be possible to verify that the output of each stage of the design flow faithfully implements the input to that stage through the use of formal methods such as model checking. However, such static analysis suffers from the problem of false positives, and a complete security analysis of a complex tool chain is not possible with current technology, owing to the exponential explosion in the number of states that must be checked.

An alternative is to build a custom set of trusted tools for security-critical hardware. This tool chain would implement a subset of the commercial tool chain's optimization functions, and the resulting designs would likely sacrifice some measure of performance for additional security. Existing research on trusted compilers could be exploited to minimize the development effort. A critical function of life-cycle protection is to ensure that the output (and transitively the input) does not contain malicious artifacts.[8] Testing can also help ensure fidelity to requirements and common failure modes. For example, it should consider the location of the system's entry points, its dependencies, and its behavior during failure.

Life-cycle management also includes delivery and maintenance. Trusted delivery ensures that the FPGA has not been tampered with from manufacturing to customer delivery. For an FPGA, maintenance includes updates to the configuration, which can occur remotely on some FPGAs. For example, a vendor might release an improved version of the bitstream that fixes bugs in the earlier version.

Secure architecture

Programmability of FPGAs is a major advantage for providing on-chip security, but this malleability introduces unique vulnerabilities. Industry is reluctant to add security features to ASICs, because the edit-compile-run cycle cost can be prohibitive. FPGAs, on the other hand, provide the opportunity to incorporate self-protective security mechanisms at a far lower cost.

Memory protection. One example of a runtime security mechanism we can build into reconfigurable hardware is memory protection. On most embedded devices, memory is flat and unprotected. A reference monitor, a well-understood concept from computer security, can enforce a policy that specifies the legal sharing of memory (and other computing resources) among cores on a chip.[9] A *reference monitor* is an access control mechanism that possesses three properties: it is self protecting, its enforcement mechanisms cannot be bypassed, and it can be subjected to analysis that ensures its correctness and completeness.[10] Reference monitors are useful in composing systems

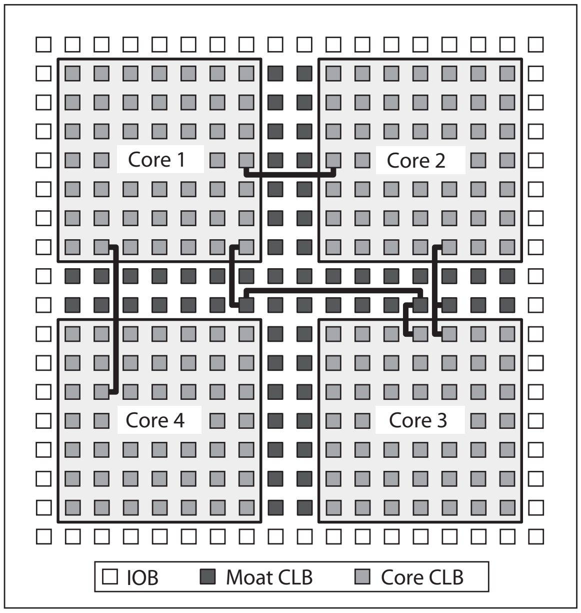

Figure 3: Sample layout for a design with four cores and a moat size of two. There are several different drawbridge configurations between the cores. (IOB: I/O block; CLB: configuration logic block.)

because they are small and do not require any knowledge of a core's inner workings.

Spatial isolation. Although synthesis tools can generate designs in which the cores are intertwined, increasing the possibility of interference, FPGAs provide a powerful means of isolation. Because applications are mapped spatially to the device, we can isolate computation resources such as cores in space by controlling the layout function,[11] as Figure 3 shows. A side benefit of the use of physical isolation of components is that it more cleanly modularizes the system. Checks for the design's correctness are easier because all parts of the chip that are not relevant to the component under test can be masked.

McLean and Moore provide similar concurrent work.[12] Although they do not provide extensive details, they appear to be using a similar technique to isolate regions of the chip by placing a buffer between them, which they call a *fence.*

Tags. As opposed to explicitly monitoring attempts to access memory, the ability to track information and its transformation as it flows through a system is a useful primitive for composing secure systems. A tag is metadata that can be attached to individual pieces of system data. Tags can be used as security labels, and, thanks to their flexibility, they can tag data in an FPGA at different granularities.

For example, a tag can be associated with an individual bit, byte, word, or larger data chunk. Once this data is tagged, static analysis can be used to test that tags are tightly bound to data and are immutable within a given program. Although techniques currently exist to enhance general-purpose processor with tags such that only the most privileged software level can add or change a tag, automatic methods of adding tags to other types of cores are needed for tags to be useful as a runtime protection mechanism. Early experiments in tagged architectures should be carefully assessed to avoid previous pitfalls.[13]

Secure communication. To get work done, cores must communicate with one another and therefore cannot be completely isolated. Current cores can communicate via either shared memory, direct connections, or a shared bus. (RF communication might be possible in the future). For communication via shared memory, the reference monitor can enforce the security policy as a function of its ability to control access to memory in general. For communication via direct connections, static analysis can verify that only specified direct connections are permitted, as we discussed earlier. Such interconnect-tracing techniques can be applied at both the device and board levels.

Communication via a shared bus must address several threats. If traffic sent over the bus is encrypted, snooping is not a problem. To address covert channels resulting from bus contention, every core can be given a fixed slice of time to use the bus. Although various optimizations have been proposed, this type of multiplexing is clearly inefficient, because a core's needs can change over time.

Future work

Embedded devices perform a critical role in both the commercial and military sectors. Increasingly more functionality is being packed onto a single device to realize the cost savings of increased integration, yet researchers have yet to address on-chip security. FPGAs can have multiple cores on the same device operating at different trust levels. Because FPGAs are at the heart of many embedded devices, new efficient security primitives are needed. We see opportunities for future work in multicore systems, further integration of our security primitives, reconfigurable updates, and both covert and side channels.

Multicore systems

As computing changes from a general-purpose uniprocessor model with disk and virtual memory to a model in which embedded devices such as cell phones perform more computing tasks, a new approach to system development is needed. Most future systems will likely be chip multiprocessor systems running multiple threads, SoCs with multiple special-purpose cores on a single ASIC, or a compromise between those two extremes on an FPGA. When the number of cores becomes large, communication between the cores over a single shared bus is impractical, and the use of direct connections (such as grid or mesh communication) becomes necessary. New techniques are necessary to mediate secure, efficient communication of multiple cores on a single chip. System design under this new model will require changes to the way in which implementations are developed to ensure performance, correctness, and security.

Further integration of security primitives

Our recent work has shown that by physically locating computations in different chip regions, and by validating the hardware boundaries between these regions, an efficient new mechanism for ensuring isolation is possible.[11] However, if a computing resource, such as an encryption unit, must be shared among security domains, then a temporal scheme (possibly based on data tagging) might be required. We are pursuing development of formal and practical methods that cooperatively apply spatial schemes, temporal schemes, and tagging to a design in a way that meets security requirements and minimizes overhead.

Reconfigurable updates

Many modern FPGAs can dynamically change part of their configuration at runtime. Partial reconfiguration makes it possible to update the circuitry in a fielded device to patch errors in the design, provide a more efficient version, change algorithm parameters, or add new data sets (such as Snort IDS rules). The avionics industry, for example, would like the ability to update systems in flight as a fault tolerance measure. Also, some supercomputers have partially reconfigurable coprocessors.

A dynamic system is more complicated and difficult to build than a static one, and this is true of the security of such a system as well. In many cases, secure state must be preserved across updates. A hot-swappable system is especially challenging because state must be transferred from the executing core to the updated core. In addition, data from the executing core must be mapped to the updated core, which might need to store the same data in a completely different location as the previous version.

In fact, the practical difficulties of implementing systems that employ partial reconfiguration has prevented its widespread use. The costs of dealing with these complexities are rarely worth the savings in on-chip area, which doubles every year anyway. However, practitioners should understand the security implications of partial reconfiguration as it applies to dynamic updates. For example, and updated core might have different security properties than the previous core. We are investigating the requirements for partial reconfiguration within our security architecture.

Channels and information leakage

Even if cores are spatially isolated, they might still be able to communicate through a covert channel. In a covert-channel attack, a *high core* leaks classified data to a *low core* that is not authorized to access classified data directly. The high source is also constrained by rules that prevent it from writing directly to the low destination. A covert channel is typically exploited by encoding data into a shared resource's observable state, such as disk usage, error conditions, or processor activity. Classical covert-channel analysis involves enumerating all shared resource and metadata on chip, identifying the share points, determining if the shared resource is exploitable, determining the bandwidth of the covert channel, and determining whether remedial action can be taken.

A side channel is slightly different from a covert channel in that the recipient is an entity outside the system that observes benign processing and can infer secrets from those observations. An example of a side-channel attack is the power-analysis attack, in which the power consumption of a crypto core is externally observed to extract the cryptographic keys. Finally, there are overt illegal channels (such as direct channels or trap doors). An example of a direct channel is a system that lacks memory protection. A core can transmit data to another core simply by copying it into a memory buffer.

Clearly, new techniques are necessary to address the problem of covert, side, and direct channels in embedded systems. In theory, a design could be statically analyzed to detect the presence of possible unintended information flows, although the scalability of this approach runs into computability limits. We are continuing to investigate solutions to this problem.

We have described a security architecture and a set of static and runtime primitives that work together to separate cores so that they do not interfere with one another, but this is only part of the picture. A successful approach must combine life-cycle management and a coherent security architecture for policy enforcement. The security architecture we describe here uses a set of primitives that complement one another, including a reference monitor for memory protection and a separation strategy that uses spatial isolation and interconnect tracing.

Designing any trustworthy complex system is challenging, and given the relative immaturity of current FPGA design approaches in which multiple computational cores from different sources are combined using commercial tools, the current state of embedded-systems security leaves much to be desired. Industry and its customers can no longer take hardware security for granted. Clearly, embedded-design practitioners must become acquainted with these problems and with related new developments from the computer security research field, such as the security primitives we've described here. Practitioners must also adapt the rich body of life-cycle management tools and techniques that have been created for trustworthy software development and apply them to hardware design. A path toward ensuring the security of the tools and the resulting product is necessary to provide accountability throughout the development process. The holistic approach to system design we've described here is a significant step in that direction.

Acknowledgments

We thank the anonymous reviewers for their comments. This research was funded in part by National Science Foundation grant CNS-0524771 and NSF Career grant CCF-0448654.

References

1. K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, vol. 34, no. 2, June 2002, pp. 171-210.

2. P. Schaumont et al., "A Quick Safari through the Reconfiguration Jungle," *Proc. 38th Design Automation Conf.* (DAC 01), ACM Press, 2001, pp. 172-177.

3. D. McGrath, "Gartner Dataquest Analyst Gives ASIC, FPGA Markets Clean Bill of Health," *EE Times*, 13 June 2005, http://www.eetimes.com/news/latest/archive/?archiveDate=06/18/2005.

4. R. Milanowski and M. Maurer, "Outsourcing Poses Unique Challenges for

the U.S. Military-Electronics Community," *Chip Design*, Aug./Setp. 2006, http://www.chipdesignmag.com/display.php?articleId=752&issueId=18.

5. T. Huffmire et al., "Designing Secure Systems on Reconfigurable Hardware," *ACM Trans. Design Automation of Electronic Systems* (TODAES), vol. 13, no. 3, July 2008, article 44.
6. C. Irvine and T. Levin, "A Cautionary Note Regarding the Data Integrity Capacity of Certain Secure Systems," *Integrity, Internal Control and Security in Information Systems*, M. Gertz, E. Guldentops, & L. Strous, eds., Kluwer Academic Publishers, 2002, pp. 3-25.
7. S. Trimberer, "Trusted Design in FPGAs," *Proc. 44th Design Automation Conf.* (DAC 07), ACM Press, 2007, pp. 5-8.
8. E.A. Anderson, C.E. Irvine, and R.R. Schell, "Subversion as a Threat in Information Warfare," *J. Information Warfare*, vol. 3, no. 2, June 2004, pp. 52-65.
9. T. Huffmire et al., "Policy-Driven Memory Protection for Reconfigurable Hardware," *Proc. 11th European Symp. Research in Computer Security* (ESORICS 06), LNCS 4189, Springer Berlin, 2006, pp. 461-478.
10. J. Anderson, *Computer Security Technology Planning Study*, tech. report ESD-TR-73-51, Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Oct. 1972.
11. T. Huffmire et al., "Moats and Drawbridges: An Isolation Primitive for Reconfigurable Hardware Based Systems," *Proc. 2007 IEEE Symp. Security and Privacy* (SP 07), IEEE CS Press, 2007, pp. 281-285.
12. M. McLean and J. Moore, "FPGA-Based Single Chip Cryptographic Solution," *Military Embedded Systems*, Mar. 2007, http://www.mil-embedded.com/articles/id/?2069.
13. H. Levy, *Capability-Based Computer Systems*, Butterworth-Heinemann, 1984.

Ted Huffmire is an assistant professor of computer science at the Naval Postgraduate School in Monterey, California. His research spans both computer security and computer architecture, specifically hardware-assisted security and the development of policy-driven mechanisms for special-purpose devices. He has a PhD in computer science from the University of California, Santa Barbara. He is a member of the IEEE.

Brett Brotherton is a senior engineer at Special Technologies Laboratory in Santa Barbara, California. His research interests include secure hardware design on FPGAs and embedded-systems design. He has an MS in computer engineering from the University of California, Santa Barbara.

Timothy Sherwood is an associate professor in the Department of Computer Science at the University of California, Santa Barbara. His research interests include computer architecture, specifically in the development of novel high-throughput methods by which systems can be constructed, monitored, and analyzed. He has a PhD in computer science and engineering from the University of California, San Diego. He is a member of the IEEE and the ACM.

Ryan Kastner is an associate professor in the Department of Computer Science and Engineering at the University of California, San Diego. His research interests focus on many aspects of embedded computing systems, including reconfigurable architectures, digital-signal processing, and security. He has a PhD in computer science from the University of California, Los Angeles.

Timothy Levin is an associate research professor at the Naval Postgraduate School in Monterey, California. His research interests include design, analysis and verification of high-assurance security architectures and dynamic security policies. He has a BS in computer science from the University of California, Santa Cruz. He is a member of the IEEE and the ACM.

Thuy D. Nguyen is a senior researcher of computer science at the Naval Postgraduate School in Monterey, California. Her research interests include high-assurance platforms, trusted operating systems, dynamic security services, multilevel security, security evaluation, and security requirements engineering. She has a BA in computer science from the University of California, Santa Cruz.

Cynthia Irvine is the director of the Center for Information Systems Security and Research (CISR) and a computer science professor at the Naval Postgraduate School in Monterey, California. Her research interest include high-assurance security. She has a PhD in astronomy from Case Western Reserve University. She is a member of the IEEE, the ACM, and the Astronomical Society of the Pacific.